Static Program Analysis using Abstract Interpretation

Arnaud Venet
venet@email.arc.nasa.gov

Guillaume Brat
brat@email.arc.nasa.gov

Kestrel Technology
NASA Ames Research Center
Moffett Field, CA 94035

Introduction

Static Program Analysis

Static program analysis consists of automatically discovering properties of a program that hold for all possible execution paths of the program.

Static program analysis is **not**

- Testing: manually checking a property for some execution paths
- Model checking: automatically checking a property for all execution paths

Program Analysis for what?

- Optimizing compilers
- Program understanding
- Semantic preprocessing:
  - Model checking
  - Automated test generation
- **Program verification**

Program Verification

- Check that every operation of a program will never cause an error (division by zero, buffer overrun, deadlock, etc.)
- Example:

```
                  int a[1000];
                  for (i = 0; i < 1000; i++) {
safe operation  ──→ a[i] = … ;   // 0 <= i <= 999
                  }
buffer overrun ──→ a[i] = … ;     // i = 1000;
```

Incompleteness of Program Analysis

- Discovering a sufficient set of properties for checking every operation of a program is an undecidable problem!
- **False positives:** operations that are safe in reality but which cannot be decided safe or unsafe from the properties inferred by static analysis.

Precision versus Efficiency

Precision: number of program operations that can be decided safe or unsafe by an analyzer.

- Precision and computational complexity are strongly related
- Tradeoff precision/efficiency: limit in the average precision and scalability of a given analyzer
- Greater precision and scalability is achieved through **specialization**

Specialization

- Tailoring the program analyzer algorithms for a specific class of programs (flight control commands, digital signal processing, etc.)
- Precision and scalability is guaranteed for this class of programs only
- Requires a lot of try-and-test to fine-tune the algorithms
- **Need for an open architecture**

Soundness

- What guarantees the soundness of the analyzer results?
- In dataflow analysis and type inference the soundness proof of the resolution algorithm is independent from the analysis specification
- An independent soundness proof precludes the use of test-and-try techniques
- **Need for analyzers correct by construction**

Abstract Interpretation

- A general methodology for designing static program analyzers that are:
  - Correct by construction
  - Generic
  - Easy to fine-tune
- Scalability is difficult to achieve but the payoff is worth the effort!

Approximation

The core idea of Abstract Interpretation is the formalization of the notion of **approximation**

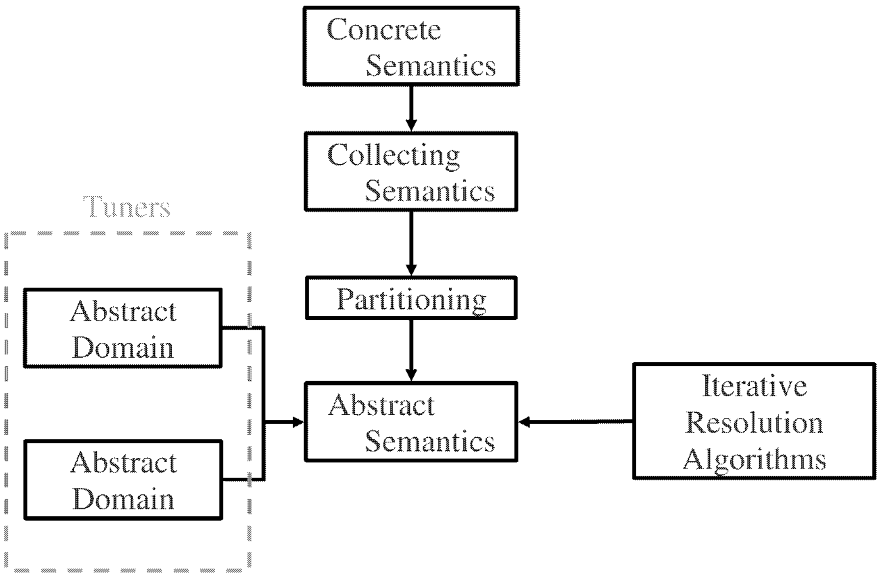
- An approximation of memory configurations is first defined
- Then the approximation of all atomic operations
- The approximation is automatically lifted to the whole program structure
- The approximation is generally a scheme that depends on some other parameter approximations

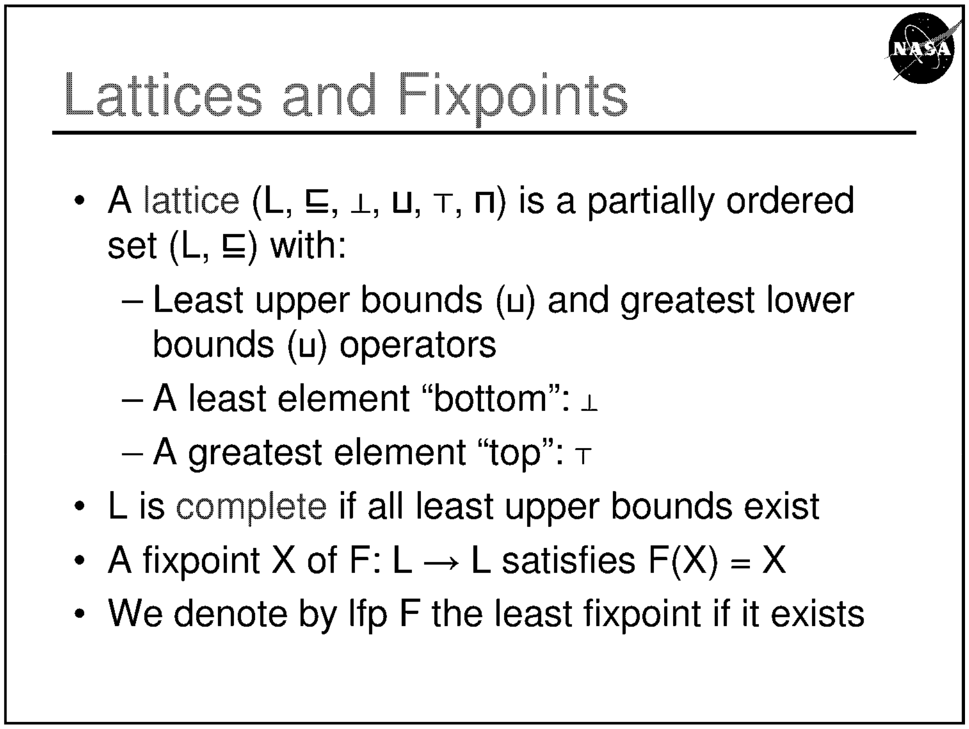
Overview of Abstract Interpretation

- Start with a formal specification of the program semantics (the concrete semantics)
- Construct abstract semantic equations w.r.t. a parametric approximation scheme
- Use general algorithms to solve the abstract semantic equations
- Try-and-test various instantiations of the approximation scheme in order to find the best fit

The Methodology of Abstract Interpretation

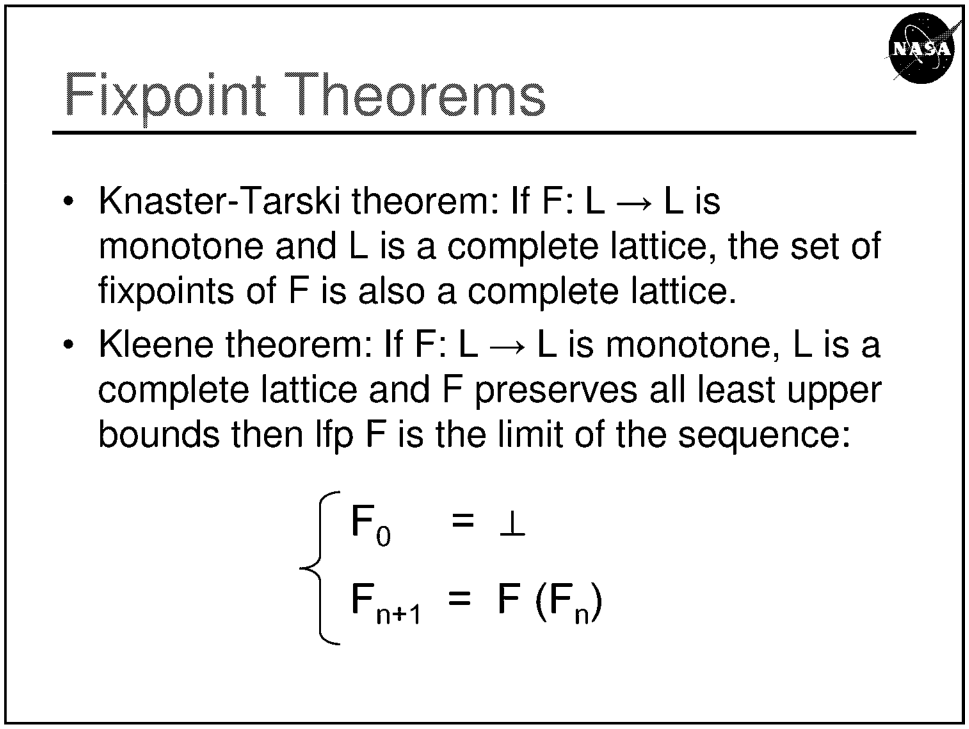
Methodology





Lattices and Fixpoints

- A lattice $(L, \sqsubseteq, \perp, \sqcup, \top, \sqcap)$ is a partially ordered set $(L, \sqsubseteq)$ with:
  - Least upper bounds ($\sqcup$) and greatest lower bounds ($\sqcup$) operators
  - A least element "bottom": $\perp$
  - A greatest element "top": $\top$
- L is complete if all least upper bounds exist
- A fixpoint X of $F: L \rightarrow L$ satisfies $F(X) = X$
- We denote by lfp F the least fixpoint if it exists

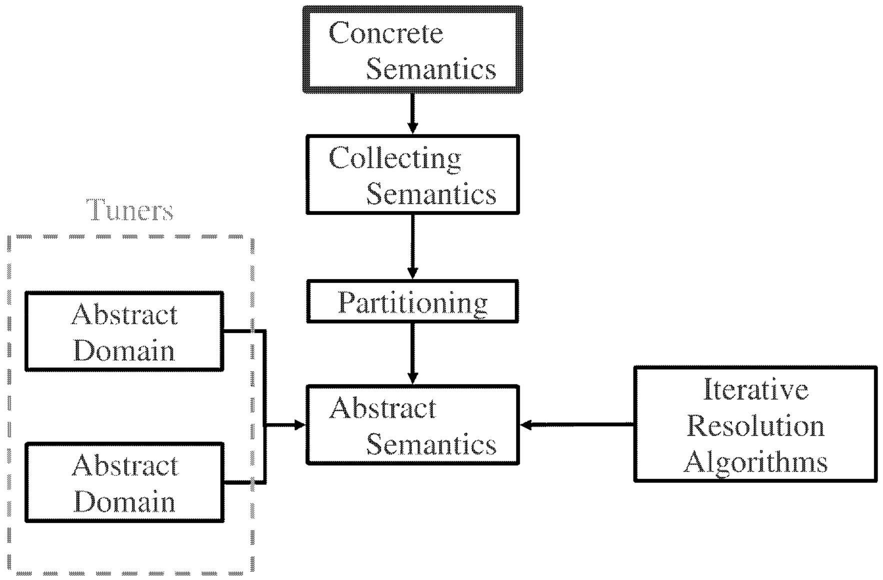


Fixpoint Theorems

- Knaster-Tarski theorem: If $F: L \rightarrow L$ is monotone and L is a complete lattice, the set of fixpoints of F is also a complete lattice.
- Kleene theorem: If $F: L \rightarrow L$ is monotone, L is a complete lattice and F preserves all least upper bounds then lfp F is the limit of the sequence:

$$\begin{cases} F_0 = \perp \\ F_{n+1} = F(F_n) \end{cases}$$

Methodology



Concrete Semantics

Small-step operational semantics: $(\Sigma, \rightarrow)$

$\boxed{s = \langle \boxed{\text{program point}}, \boxed{\text{env}} \rangle}$ $\boxed{s \rightarrow s'}$

<u>Example:</u>

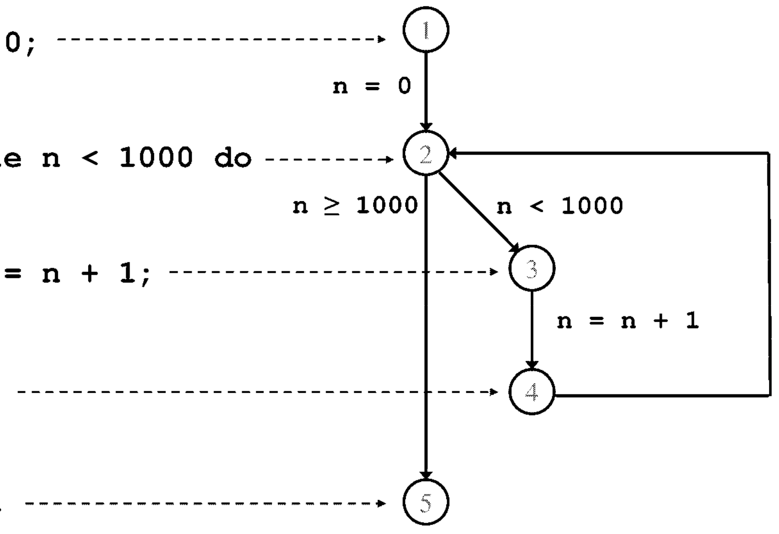
```
1:  n = 0;
2:  while n < 1000 do
3:    n = n + 1;
4:  end
5:  exit
```

$\langle 1, n \Rightarrow \Omega \rangle \rightarrow \langle 2, n \Rightarrow 0 \rangle \rightarrow \langle 3, n \Rightarrow 0 \rangle \rightarrow \langle 4, n \Rightarrow 1 \rangle$
$\rightarrow \langle 2, n \Rightarrow 1 \rangle \rightarrow \ldots \rightarrow \langle 5, n \Rightarrow 1000 \rangle$

Undefined value

Control Flow Graph

```
1:  n = 0;
2:  while n < 1000 do
3:    n = n + 1;
4:  end
5:  exit
```



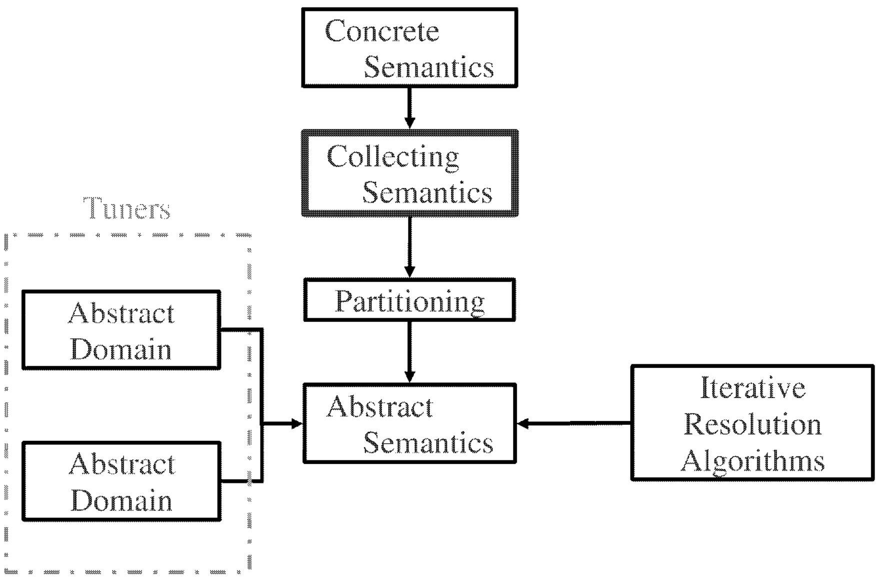
Transition Relation

Control flow graph: $\textcircled{i} \xrightarrow{\text{op}} \textcircled{j}$

Operational semantics: $\langle \textcircled{i}, \varepsilon \rangle \rightarrow \langle \textcircled{j}, [\![\text{op}]\!]\ \varepsilon \rangle$

Semantics of op

Methodology



Collecting Semantics

The collecting semantics is the set of observable behaviours in the operational semantics. It is the starting point of any analysis design.

- The set of all descendants of the initial state
- The set of all descendants of the initial state that can reach a final state
- The set of all finite traces from the initial state
- The set of all finite and infinite traces from the initial state
- etc.

Which Collecting Semantics?

- Buffer overrun, division by zero, arithmetic overflows: state properties
- Deadlocks, un-initialized variables: finite trace properties
- Loop termination: finite and infinite trace properties

State properties

The set of descendants of the initial state s_0:

$$\boxed{S = \{s \mid s_0 \to \ldots \to s\}}$$

<u>Theorem:</u> $F : (\wp(\Sigma), \subseteq) \to (\wp(\Sigma), \subseteq)$

$$F(S) = \{s_0\} \cup \{s' \mid \exists s \in S: s \to s'\}$$

$$\boxed{S = \mathrm{lfp}\ F}$$

Example

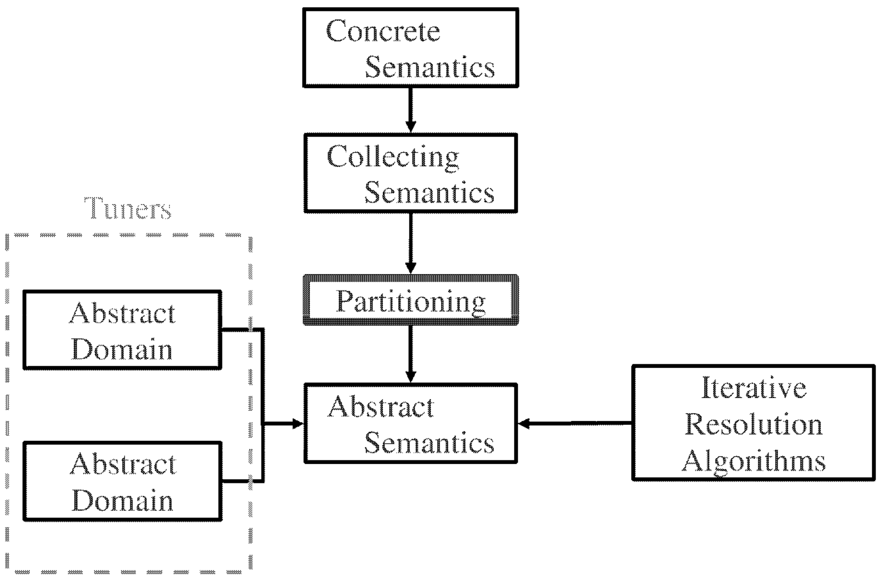
```
1:  n = 0;
2:  while n < 1000 do
3:    n = n + 1;
4:  end
5:  exit
```

$$S = \{\langle 1, n \Rightarrow \Omega\rangle, \langle 2, n \Rightarrow 0\rangle, \langle 3, n \Rightarrow 0\rangle, \langle 4, n \Rightarrow 1\rangle, \langle 2, n \Rightarrow 1\rangle, ..., \langle 5, n \Rightarrow 1000\rangle\}$$

Computation

- $F0 = \varnothing$
- $F1 = \{\langle 1,n\Rightarrow\Omega\rangle \}$
- $F2 = \{\langle 1,n\Rightarrow\Omega\rangle, \langle 2,n\Rightarrow 0\rangle \}$
- $F3 = \{\langle 1,n\Rightarrow\Omega\rangle, \langle 2,n\Rightarrow 0\rangle, \langle 3,n\Rightarrow 0\rangle \}$
- $F4 = \{\langle 1,n\Rightarrow\Omega\rangle, \langle 2,n\Rightarrow 0\rangle, \langle 3,n\Rightarrow 0\rangle, \langle 4,n\Rightarrow 1\rangle \}$
- ...

Methodology

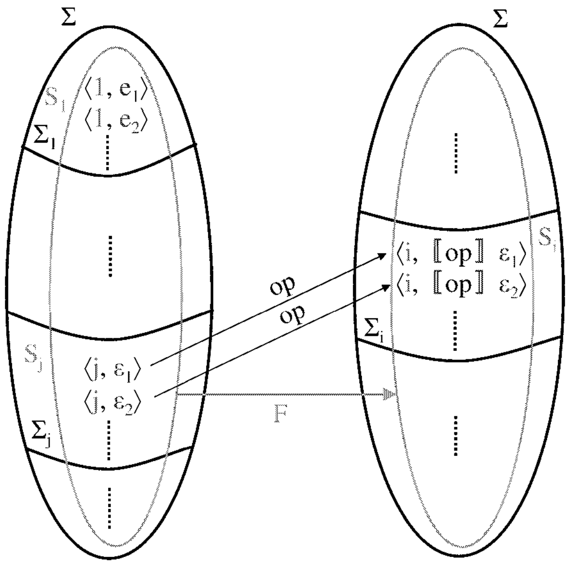


Partitioning

We partition the set S of states w.r.t. program points:

- $\Sigma = \Sigma_1 \oplus \Sigma_2 \oplus \ldots \oplus \Sigma_n$
- $\Sigma_i = \{\langle k, \varepsilon\rangle \in \Sigma \mid k = i\}$
- $F(S_1, \ldots, S_n)_i = \{s' \in S_i \mid \exists j\ \exists s \in S_j : s \to s'\}$
- $F(S_1, \ldots, S_n)_i = \{\langle i, [\![op]\!]\ \varepsilon\rangle \mid \text{ⓙ} \xrightarrow{op} \text{ⓘ} \in CFG(P)\}$
- $F(S_1, \ldots, S_n)_0 = \{ s_0 \}$

Illustration



Semantic Equations

- Notation: E_i = set of environments at program point i
- System of semantic equations:

$$\boxed{E_i = \bigcup \{ \llbracket op \rrbracket E_j \mid (j) \xrightarrow{op} (i) \in \mathrm{CFG}(P) \}}$$

- Solution of the system **= S =** lfp **F**

Example

```
1:  n = 0;
2:  while n < 1000 do
3:    n = n + 1;
4:  end
5:  exit
```

$E_1 = \{n \Rightarrow \Omega\}$

$E_2 = [\![\texttt{n = 0}]\!]\ E_1 \cup E_4$

$E_3 = E_2 \cap\ ]-\infty, 999]$

$E_4 = [\![\texttt{n = n + 1}]\!]\ E_3$

$E_5 = E_2 \cap [1000, +\infty[$

Example

```
1:  n = 0;
2:  while n < 1000 do
3:    n = n + 1;
4:  end
5:  exit
```

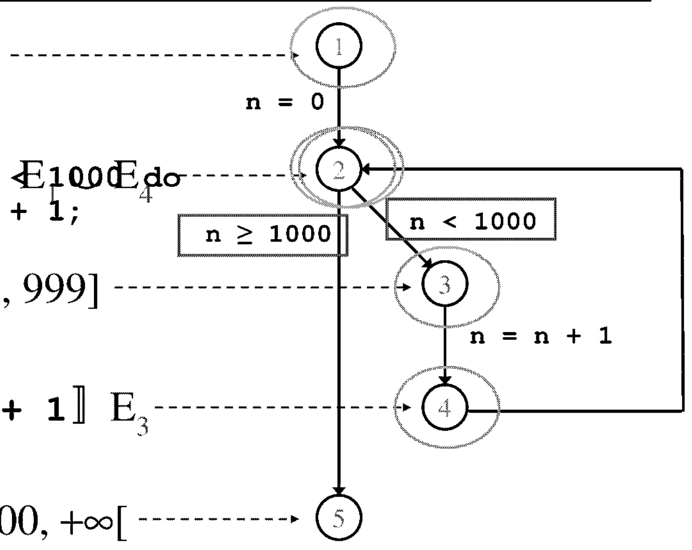
$E_1 = \{n \Rightarrow \Omega\}$

$E_2 = [\![\texttt{n = 0}]\!]\ E_1 \cup E_4$

$E_3 = E_2 \cap\ ]-\infty, 999]$

$E_4 = [\![\texttt{n = n + 1}]\!]\ E_3$

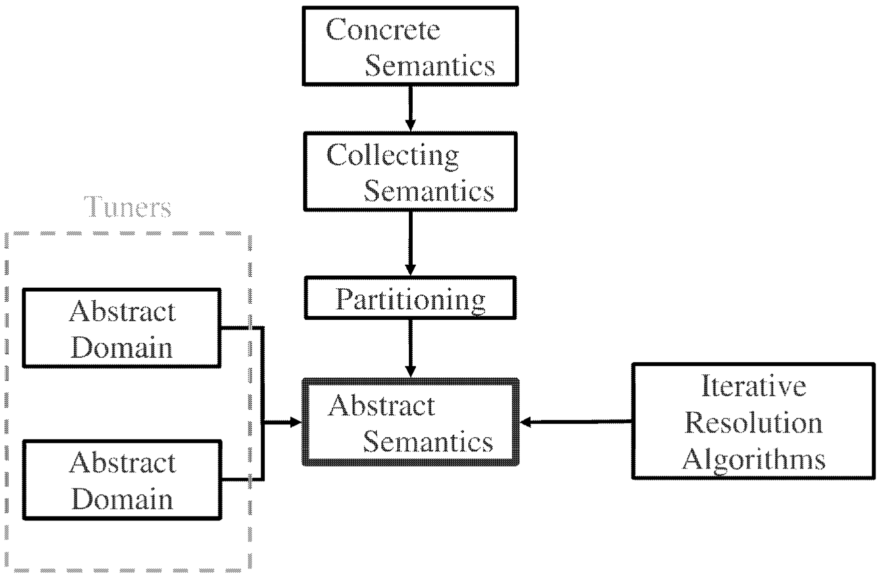
$E_5 = E_2 \cap [1000, +\infty[$



Other Kinds of Partitioning

- In the case of collecting semantics of traces:
  - Partitioning w.r.t. procedure calls: **context sensitivity**
  - Partitioning w.r.t. executions paths in a procedure: **path sensitivity**
  - Dynamic partitioning (Bourdoncle)

Methodology



Approximation

Problem: Compute a sound approximation $S^\#$ of S

$$\boxed{S \subseteq S^\#}$$

Solution: Galois connections

Galois Connection

L_1, L_2 two lattices

Abstract domain

$$\begin{tikzcd}(L_1, \subseteq) \arrow[r, shift right, "\alpha"'] & (L_2, \leq) \arrow[l, shift right, "\gamma"']\end{tikzcd}$$

- $\forall x \forall y : \alpha(x) \leq y \Leftrightarrow x \subseteq \gamma(y)$
- $\forall x \forall y : x \subseteq \gamma \circ \alpha(x) \ \& \ \alpha \circ \gamma(y) \leq y$

Fixpoint Approximation

$$\begin{tikzcd} L_2 \arrow[r, "\alpha \circ F \circ \gamma"] \arrow[d, "\gamma"'] & L_2 \\ L_1 \arrow[r, "F"'] & L_1 \arrow[u, "\alpha"'] \end{tikzcd}$$

<u>Theorem:</u>

$$\boxed{\text{lfp } F \subseteq \gamma(\text{lfp } \alpha \circ F \circ \gamma)}$$

Abstracting the Collecting Semantics

- Find a Galois connection:

$$(\wp(\Sigma), \subseteq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (\Sigma^{\#}, \leq)$$

- Find a function: $\alpha \circ F \circ \gamma \leq F^{\#}$

$$\boxed{\text{Partitioning} \Rightarrow \text{Abstract sets of environments}}$$

Abstract Algebra

- Notation: E the set of all environments
- Galois connection:

$$\begin{tikzcd}(\wp(E), \subseteq) \arrow[r, shift right, "\alpha"'] & (E^\#, \leq) \arrow[l, shift right, "\gamma"']\end{tikzcd}$$

- $\cup$, $\cap$ approximated by $\cup^\#$, $\cap^\#$
- Semantics $[\![\mathbf{op}]\!]$ approximated by $[\![\mathbf{op}]\!]^\#$

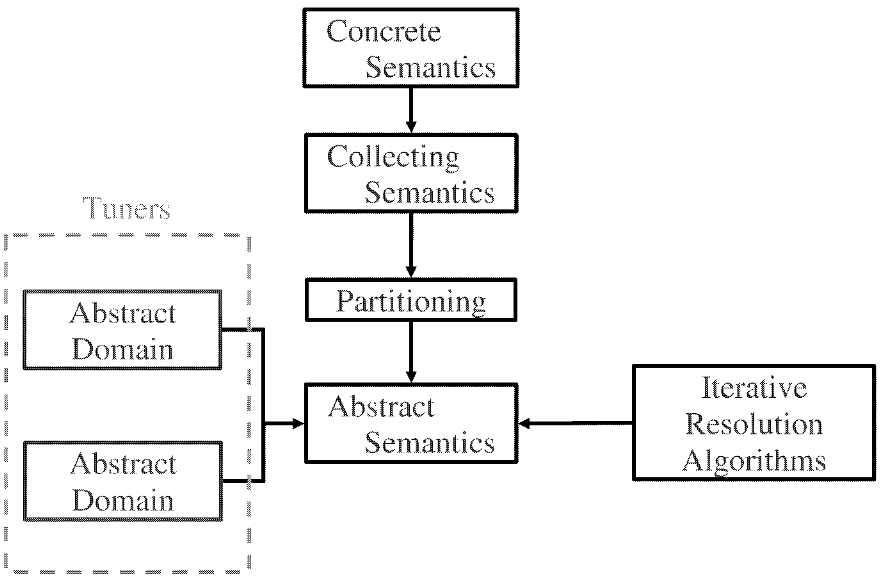
$$\boxed{\alpha \circ [\![\mathbf{op}]\!] \circ \gamma \subseteq [\![\mathbf{op}]\!]^\#}$$

Abstract Semantic Equations

```
1:  n = 0;
2:  while n < 1000 do
3:    n = n + 1;
4:  end
5:  exit
```

$$\boxed{\begin{array}{l}
E_1^\# = \alpha(\{n \Rightarrow \Omega\}) \\
E_2^\# = [\![\texttt{n = 0}]\!]^\# E_1^\# \cup^\# E_4^\# \\
E_3^\# = E_2^\# \cap^\# \alpha(]-\infty, 999]) \\
E_4^\# = [\![\texttt{n = n + 1}]\!]^\# E_3^\# \\
E_5^\# = E_2^\# \cap^\# \alpha([1000, +\infty[)
\end{array}}$$

Methodology



Abstract Domains

Environment: $x \Rightarrow v$, $y \Rightarrow w$, ...

Various kinds of approximations:

- Intervals (nonrelational):

  $x \Rightarrow [a, b]$, $y \Rightarrow [a', b']$, ...

- Polyhedra (relational):

  $x + y - 2z \leq 10$, ...

- Difference-bound matrices (weakly relational):

  $y - x \leq 5$, $z - y \leq 10$, ...

Example: intervals

```
1:  n = 0;
2:  while n < 1000 do
3:    n = n + 1;
4:  end
5:  exit
```

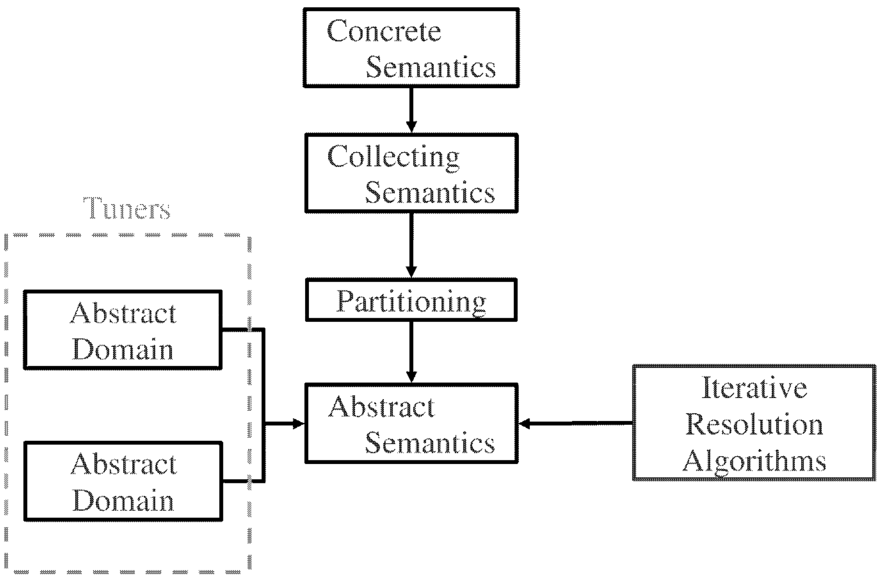
- Iteration 1: $E_2^\# = [0, 0]$
- Iteration 2: $E_2^\# = [0, 1]$
- Iteration 3: $E_2^\# = [0, 2]$
- Iteration 4: $E_2^\# = [0, 3]$
- ...

Problem

How to cope with lattices of infinite height?

Solution: automatic extrapolation operators

Methodology



Widening operator

Lattice (L, ≤): $\nabla : L \times L \to L$

- Abstract union operator:

$$\forall x \forall y : \; x \leq x \nabla y \;\; \& \;\; y \leq x \nabla y$$

- Enforces convergence: $(x_n)_{n \geq 0}$

$$\begin{cases} y_0 = x_0 \\ y_{n+1} = y_n \nabla x_{n+1} \end{cases}$$

$\boxed{(y_n)_{n\geq 0} \text{ is ultimately stationary}}$

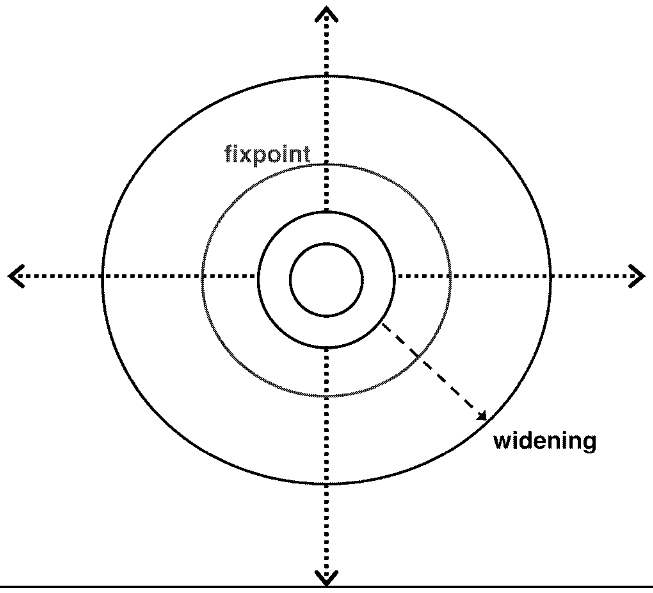
Widening of intervals

$$\boxed{[a, b] \nabla [a', b']}$$

- If $a \leq a'$ then a else $-\infty$
- If $b' \leq b$ then b else $+\infty$

↪ Open unstable bounds (jump over the fixpoint)

Widening and Fixpoint



Iteration with widening

```
1:   n = 0;
2:   while n < 1000 do
3:     n = n + 1;
4:   end
5:   exit
```

$$\boxed{(E_2^\#)_{n+1} = (E_2^\#)_n \nabla \left([\![\texttt{n = 0}]\!]^\# (E_1^\#)_n \cup^\# (E_4^\#)_n \right)}$$

Iteration 1 (union): $E_2^\# = [0, 0]$

Iteration 2 (union): $E_2^\# = [0, 1]$

Iteration 3 (widening): $E_2^\# = [0, +\infty] \Rightarrow$ stable

Imprecision at loop exit

```
1:   n = 0;
2:   while n < 1000 do
3:     n = n + 1;
4:   end
5:   ~~exit;~~ t[n] = 0; //t has 1500 elements
```

False positive!!!

- $E_5^\# = [1000, +\infty[$
- The information is present in the equations

Narrowing operator

Lattice $(L, \leq)$: $\Delta : L \times L \rightarrow L$

- Abstract intersection operator:

$$\forall x \forall y : x \cap y \leq x \,\Delta\, y$$

- Enforces convergence: $(x_n)_{n \geq 0}$

$$\begin{cases} y_0 = x_0 \\ y_{n+1} = y_n \,\Delta\, x_{n+1} \end{cases}$$

$\boxed{(y_n)_{n\geq 0} \text{ is ultimately stationary}}$

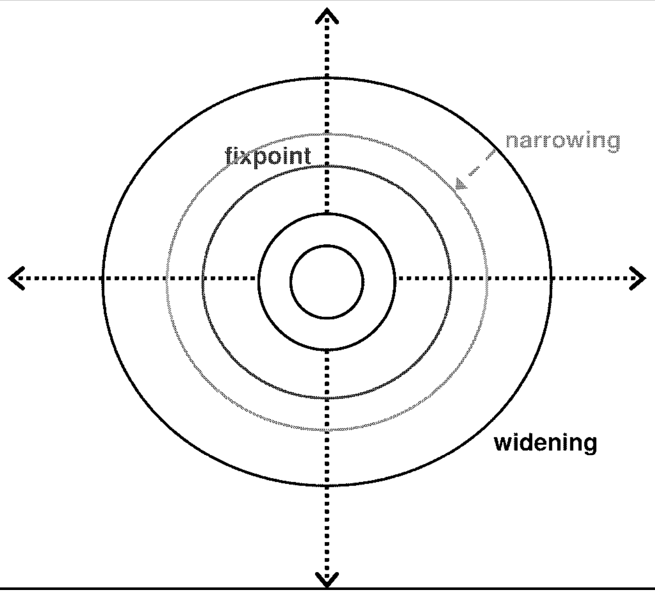
Narrowing of intervals

$\boxed{[a, b] \,\Delta\, [a', b']}$

- If $a = -\infty$ then a' else a
- If $b = +\infty$ then b' else b

↳ Refine open bounds

Narrowing and Fixpoint



Iteration with narrowing

```
1:  n = 0;
2:  while n < 1000 do
3:    n = n + 1;
4:  end
5:  exit; t[n] = 0;
```

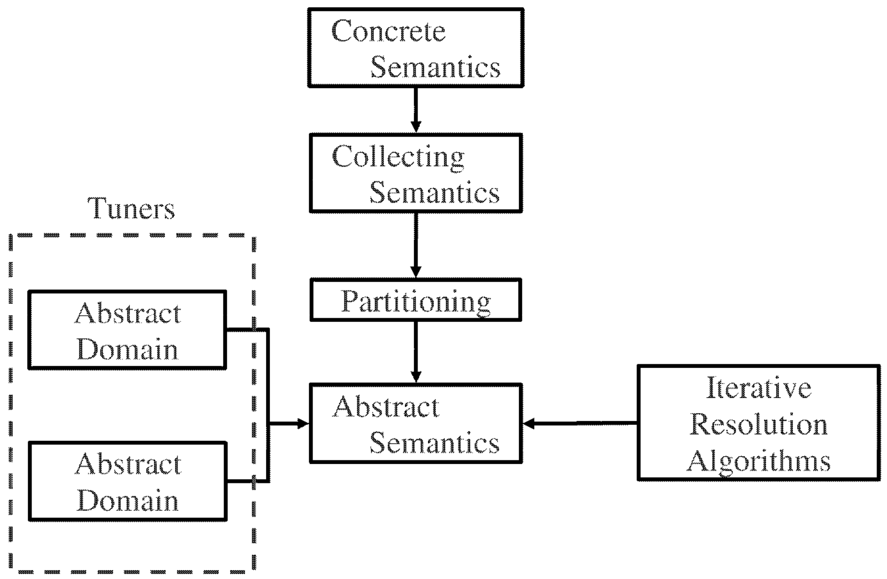
$$(E_2^{\#})_{n+1} = (E_2^{\#})_n \,\Delta \left([\![\texttt{n = 0}]\!]^{\#} (E_1^{\#})_n \cup^{\#} (E_4^{\#})_n \right)$$

Beginning of iteration: $E_2^{\#} = [0, +\infty[$

Iteration 1: $E_2^{\#} = [0, 1000] \Rightarrow$ stable

Consequence: $E_5^{\#} = [1000, 1000]$

Methodology



Tuning the abstract domains

```
n = 0;
k = 0;
while n < 1000 do
  n = n + 1;
  k = k + 1;
end
exit
```

- Intervals:

$$E_4^{\#} = \langle\, n \Rightarrow [0, 1000],\ k \Rightarrow [0, +\infty[\, \rangle$$

- Convex polyhedra or DBMs:

$$E_4^{\#} = \langle\, 0 \le n \le 1000,\ 0 \le k \le 1000,\ n - k = 0 \,\rangle$$

Comparison with Data Flow Analysis

Data Flow Framework

- Forward Data Flow Equations

$$\mathrm{in}(B) = \begin{cases} Init & ,\mathrm{B} = \mathrm{entry} \\ \bigcap\limits_{P \in \mathrm{Pred}(B)} \mathrm{F}_B(\mathrm{in}(B)) & ,\mathrm{otherwise} \end{cases}$$

- L is a lattice
- in(*B*) ∈ L is the data-flow information on entry to *B*
- *Init* is the appropriate initial value on entry to the program
- F_B is the transformation of the data-flow information upon executing block *B*
- ∩ models the effect of combining the data-flow information on the edges entering a block

Data-Flow Solutions

- Solving the data-flow equations computes the meet-over-all-paths (MOP) solution

$$\boxed{\mathrm{MOP}(B) = \bigcap_{p \in \mathrm{Path}(B)} F_p(\mathit{Init}) \text{ for } B = \mathit{entry}, B1, \ldots, Bn, \mathit{exit}}$$

- If F_B is monotone, i.e., $\boxed{F_B(x \cap y) \subseteq F_B(x) \cap F_B(y)}$
- then MOP ≤ MFP (maximum fixpoint)
- If F_B is distributive, i.e., $\boxed{F_B(x \cap y) = F_B(x) \cap F_B(y)}$
- then MOP = MFP

Typical Data-Flow Analyses

- Reaching Definitions
- Available Expressions
- Live Variables
- Upwards-Exposed Uses
- Copy-Propagation Analysis
- Constant-Propagation Analysis
- Partial-redundancy Analysis

Reaching Definitions

- Data-flow equations:

  $\forall i$: RCHin(i) = U (GEN(j) $\cup$ (RCHin(j) $\cap$ PRSV(j)))

where
  - PRSV are the definitions preserved by the block
  - GEN are the definitions generated by the blocks

- This is an iterative forward bit-vector problem
  - Iterative: it is solved by iteration from a set of initial values
  - Forward: information flows in direction of execution
  - Bit-vector: each definition is represented as a 1 (may reach given point) or a 0 (it does not reach this point)

AI versus classical DFA

- Classical DFA is stated in terms of properties whereas AI is usually stated in terms of models, whence the duality in the formulation.
- In classical DFA the proof of soundness must be made separately whereas it comes from the construction of the analysis in AI.
- Added benefits of AI:
  - Approximation of fixpoints in AI
    - Widening operators
    - Narrowing operators
  - Abstraction is explicit in AI
    - Galois connections
    - Can build a complex analysis as combination of basic, already-proved-correct, analyses

Annotated Bibliography

References

- The historic paper:
  - Patrick Cousot & Radhia Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 238—252, Los Angeles, California, 1977. ACM Press, New York, NY, USA.
- Accessible introductions to the theory:
  - Patrick Cousot. Semantic foundations of program analysis. In S.S. Muchnick and N.D. Jones, editors, *Program Flow Analysis: Theory and Applications,* Ch. 10, pages 303—342, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, U.S.A., 1981.
  - Patrick Cousot & Radhia Cousot. Abstract interpretation and application to logic programs. *Journal of Logic Programming,* 13(2—3):103—179, 1992.
- Beyond Galois connections, a presentation of relaxed frameworks:
  - Patrick Cousot & Radhia Cousot. Abstract interpretation frameworks. *Journal of Logic and Computation,* 2(4):511—547, August 1992.
- A thorough description of a static analyzer with all the proofs (difficult to read):
  - Patrick Cousot. The Calculational Design of a Generic Abstract Interpreter. Course notes for the NATO International Summer School 1998 on Calculational System Design. Marktoberdorf, Germany, 28 July—9 august 1998, organized by F.L. Bauer, M. Broy, E.W. Dijkstra, D. Gries and C.A.R. Hoare.

References

- The abstract domain of intervals:
  - Patrick Cousot & Radhia Cousot. Static Determination of Dynamic Properties of Programs. In B. Robinet, editor, *Proceedings of the second international symposium on Programming,* Paris, France, pages 106—130, april 13-15 1976, Dunod, Paris.

- The abstract domain of convex polyhedra:
  - Patrick Cousot & Nicolas Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Conference R ecord of the Fifth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Program ming Languages*, pages 84—97, Tucson, Arizona, 1978. ACM Press, New York, NY, USA.

- Weakly relational abstract domains:
  - Antoine Miné. The Octagon Abstract Domain. In Analysis, Slicing and Transformation (part of Working Conference on Reverse Engineering), October 2001, IEEE, pages 310-319.
  - Antoine Miné. A New Numerical Abstract Domain Based on Difference-Bound Matrices. In Program As Data Objects II, May 2001, LNCS 2053, pages 155-172.

- Classical data flow analysis:
  - Steven Muchnick. *Advanced Compiler Design and Implementation.* Morgan Kaufmann, 1997.